# HARP: Energy-Aware and Adaptive Management of Heterogeneous Processors

Till Smejkal[*]
TU Dresden
Dresden, Germany
till.smejkal@tu-dresden.de

Robert Khasanov[*]
TU Dresden
Dresden, Germany
robert.khasanov@tu-dresden.de

Jeronimo Castrillon
TU Dresden
Dresden, Germany
jeronimo.castrillon@tu-dresden.de

Hermann Härtig
TU Dresden
Dresden, Germany
hermann.haertig@tu-dresden.de

## Abstract

Energy efficiency has become a key concern in modern computing. Major processor vendors now offer single-ISA heterogeneous processors that combine powerful and energy-efficient cores, such as Arm's big.LITTLE CPUs, Apple's M-series chips, and Intel P/E systems. However, today's OS schedulers, relying on simple cost-based thread allocation strategies, fail to fully exploit their potential.

This paper presents HARP, a Linux-integrated resource-management framework for heterogeneous processors. HARP leverages application behavior through online monitoring or application descriptions and introduces a lightweight interface for two-way communication between applications and the resource manager. Through this interface, HARP learns application characteristics to guide allocation decisions, which are then relayed back to the applications so they can adapt accordingly. HARP supports various programming models, from OpenMP and Intel TBB to custom models with adaptivity features, significantly improving performance and energy efficiency, particularly in multi-application scenarios. On two representative heterogeneous systems, HARP reduces the average execution time by 12 % and the energy consumption by 28 % compared to existing methods. Overall, HARP marks a crucial step toward energy-efficient computing across diverse architectures.

## CCS Concepts

• **Computer systems organization → Heterogeneous (hybrid) systems**; • **Software and its engineering → Power management**.

## Keywords

Heterogeneous Processors, Resource Management, Energy-Efficiency

**ACM Reference Format:**
Till Smejkal, Robert Khasanov, Jeronimo Castrillon, and Hermann Härtig. 2025. HARP: Energy-Aware and Adaptive Management of Heterogeneous Processors. In *26th ACM Middleware Conference (Middleware '25), December 15–19, 2025, Nashville, TN, USA*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3721462.3770774

[*]Equal contribution from the authors.

## 1 Introduction

With the introduction of Intel Alder Lake processors [28], and AMD's Phoenix 2 and Strix Point processors [24, 37], the trend of heterogeneous CPUs — well-known from Arm big.LITTLE [20] and Apple M-series [3] — has reached the domain of x86 desktop and server computers. All major processor vendors thus include single-ISA heterogeneous processors in their portfolios.

Heterogeneous CPUs typically combine high-performance cores (named *big* or *P-cores*) with energy-efficient ones (*LITTLE* or *E-cores*). The former offer better single-thread performance but with higher power consumption, while the latter deliver lower performance but substantially reduce power usage. This shift away from homogeneous processor designs requires revisiting resource management strategies in a modern operating system (OS). Heterogeneous CPUs provide a trade-off for processor management to the OS. Application threads can either be assigned to high-performance cores for better performance or to energy-efficient cores, which deliver lower performance but conserve total system energy. However, the extent to which an application benefits from high-performance cores depends on its characteristics; for example, for memory-bound and I/O-bound applications, the performance difference between core types can be negligible.

Modern OSes have already begun integrating mechanisms to manage resources on heterogeneous CPUs more effectively. Examples include the Intel Thread Director support in Windows 11 [16] and Linux's Energy-Aware-Scheduler (EAS) for Arm big.LITTLE systems [47], both of which improve resource allocation, enhancing system performance and energy efficiency. However, two main features are missing in state-of-the-art heterogeneity-aware resource management in OSes (Section 2).

First, resource management is focused primarily on thread-to-core pinning, considering only individual thread behavior rather than that of the entire application. This approach overlooks the interactions between threads within a program, especially on heterogeneous CPUs where the assignment of a single thread may affect the performance of all other threads. This is particularly evident in cases where load imbalances between threads arise and appropriate load re-balancing is more complicated on heterogeneous processors than on homogeneous ones [57]. Second, the resource-management decisions are not communicated to the applications, nor can the applications influence these decisions, aside from setting thread affinity masks to exclude certain cores. Consequently, once an application has started, it relies solely on the OS for resource-management decisions. The missing communication channel prevents applications from, e. g., adapting their execution to react to changes in resource availability during runtime.

This paper proposes HARP, a generic runtime resource management system (Section 4) that provides a unified resource-allocation solution across desktop systems, servers, and mobile devices with heterogeneous processors. Instead of individual thread-to-core pinning, HARP jointly manages all application threads; it leverages application heterogeneity-aware behavior characteristics obtained either from online monitoring (Section 5) or sophisticated offline analyses to achieve efficient resource utilization. Furthermore, HARP introduces an interface between applications and the resource manager (RM) to communicate RM decisions to applications, allowing applications to internally adapt to the assigned resources and thus improve their efficiency. By combining heterogeneity-aware application behavior characteristics with internal application adaptation, HARP addresses key gaps in modern OS design.

Our evaluation (Section 6) shows that HARP reduces energy consumption by an average of 28 % and improves the execution time by 12 % on the two analyzed systems, the Arm Odroid XU3-E and the Intel Raptor Lake Core i9-13900K. To our knowledge, HARP is the first system capable of automatically managing applications with dynamic properties on heterogeneous processors.

## 2 The Challenge of Managing Heterogeneous Processors

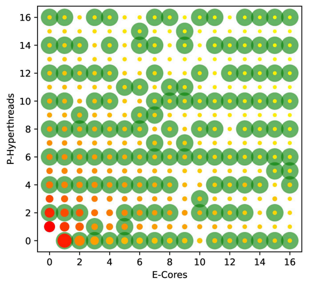
Efficient processor management is an important task of the OS, significantly impacting overall system performance. Processor management typically involves three key aspects: **(1)** controlling core idle states, **(2)** selecting clock frequencies and voltages, and **(3)** assigning application threads to cores. Given the performance-critical role within the OS, these management decisions have to be fast and efficient, which is why modern OSes rely on heuristics.

With the introduction of heterogeneous processors, which combine different types of cores on a single die, processor management needs to be revisited. While idle state selection as well as clock frequency control are not majorly affected by heterogeneous processor designs, new challenges arise, for the thread-to-core assignment. This section outlines two major challenges that must be addressed to fully utilize heterogeneous processors: accounting for application behavior and enabling dynamic application adaptation.

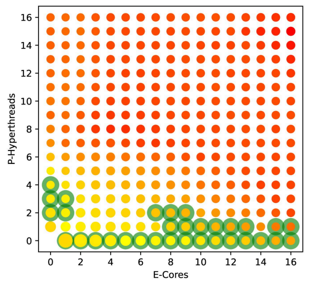
### 2.1 Unique Application Behavior

As mentioned earlier, heterogeneous processors combine high-performance cores with energy-efficient ones, offering a trade-off between performance and energy consumption. This creates a challenge for resource management, as different applications benefit from different core types and their combinations. Thus, optimal core allocation must account for the specific characteristics of each application to ensure overall performance and energy efficiency.

Figure 1 illustrates this variability for two NAS Parallel Benchmark applications running on an Intel Raptor Lake Core i9-13900K. Each dot represents an application configuration with its thread distribution across E-cores (x-axis) and P-Hyperthreads (y-axis). Dot size reflects execution time, and color indicates energy consumption (red for higher consumption). Pareto-optimal configurations were identified using four objectives: execution time, energy consumption, number of P-cores, and number of E-cores (all minimized). These Pareto-optimal points are highlighted in green.



**(a) ep.C**



**(b) mg.C**

**Figure 1: Performance and energy consumption of two applications on an Intel Raptor Lake Core i9-13900K with varying configurations.**

For `ep.C` (Figure 1a), performance and energy scale smoothly with increasing core counts, particularly towards the upper right corner, indicating benefits from using both P-cores and E-cores. Interestingly, the Pareto front for `ep.C` favors even numbers of P-hyperthreads, as this benchmark achieves better performance when both hyperthreads of a P-core are utilized. In contrast, `mg.C` (Figure 1b) does not benefit from more resources, especially when they are heterogeneous core combinations. As shown in the figure, the energy consumption increases with higher core counts without reducing the execution time. Instead, `mg.C` performs much better when primarily scheduled on energy-efficient cores. An RM targeting heterogeneous cores needs to integrate these behavioral differences in its resource allocation strategies in order to manage the system as efficient as possible.

### 2.2 Need for Dynamic Application Adaptation

Running applications in a dynamic environment poses a second challenge: applications need to adapt to changing resource allocations at runtime. For data-parallel applications like those in Figure 1, optimal performance is typically achieved when the number of threads matches the number of allocated cores. Running more threads than cores increases scheduling overhead and may cause lock-holder preemption, while running fewer threads leaves resources underutilized. Unfortunately, many scalable applications are *moldable* [18], meaning their parallelization degree is fixed at launch and cannot be adjusted dynamically. This becomes problematic in dynamic multi-application environments, such as desktop and server systems, where available resources fluctuate as applications start and terminate.

On heterogeneous CPUs, some applications benefit from more advanced adaptations beyond their parallelization degree. For example, they may need to redistribute workloads across cores, as E-cores take longer to execute and can cause P-cores to stall [54]. Applications that use pipeline parallelism can also improve when they respect performance differences during work distribution [71]. In some cases, applications may even switch algorithms based on core types [52, 53, 59]. To exploit such adaptivity features, applications must be aware of their assigned resources and capable of adjusting their internal behavior accordingly. This requires coordination between the RM and applications: the RM should dynamically inform applications about current resource allocations, while applications, in turn, should provide feedback on their behavior characteristics to improve future RM allocations. Such two-way coordination is essential to enable both system and application-level optimizations.

## 3 Resource Management in the Wild

This section reviews how OS schedulers address the challenges of heterogeneous processors, examines Hybrid Application Mapping from the embedded domain as an alternative resource-management approach, and discusses approaches incorporating in-application adaptivity. We then compare these methods with HARP, showing how it unifies their strengths while overcoming their limitations.

### 3.1 Improvements within OS Schedulers

Since the dawn of multiprogramming, OS schedulers have evolved from time-sharing a single CPU to managing resource allocations. Initially, schedulers focused on dividing CPU time among tasks. With the advent of multi-core processors, they also had to handle *space-sharing* — allocating tasks across multiple cores. This brought new challenges, such as contention for shared caches and memory controllers, which led to advanced thread placement strategies that group threads benefiting from shared resources while separating those that compete for them [10, 80].

Following this evolution, researchers began exploring single-ISA heterogeneous processors, whose benefits came with significant new scheduling challenges [2, 36, 43]. Before such processors became commercially available, studies emulated them via simulation [7, 36], frequency scaling [55, 57], or modified core configurations [34]. Many of these works assigned CPU-intensive threads to fast cores [7, 34, 36, 57], while others proposed alternative strategies like mapping sequential applications or phases to fast cores [55].

Once heterogeneous CPUs entered the market, OS vendors began integrating new strategies, following the approach of assigning CPU-intensive tasks to fast cores. For Arm's big.LITTLE processors, Linux introduced the Energy-Aware Scheduler (EAS), which uses CPU energy models to optimize task placement [47]. EAS tracks task CPU demand via Per-Entity Load Tracking (PELT), allocating tasks to minimize energy consumption, preferring LITTLE cores for low-demand tasks. PELT data is also used by the Dynamic Voltage and Frequency Scaling (DVFS) subsystem, making EAS tightly integrated with dynamic frequency scaling.

Intel's Alder Lake processors feature the Intel Thread Director (ITD), a hardware unit that monitors the instruction mix of each thread and core state at nanosecond granularity [29]. ITD uses machine learning to classify threads and calculate performance and energy-efficiency scores per core type and provides this feedback to the OS at the microsecond level, thus following an approach formerly discussed by Van et al. [69]. Windows 11 already integrates ITD [16], whereas Linux support is under development [12, 44]. In parallel, prototypes such as PMCSched demonstrate how ITD data can be incorporated into Linux task placement decisions [8, 56].

While these systems account for thread behavior on different core types, they overlook the collective impact on overall application performance. Furthermore, they do not propagate resource allocation decisions to applications, preventing them from adapting their execution. This lack of coordination between the OS scheduler and applications limits further optimization of resource utilization.

### 3.2 Leveraging Holistic Application Behaviour with Hybrid Application Mapping

To effectively manage resources on heterogeneous systems, understanding an application's overall behavior is crucial. However, performing this analysis online is often infeasible due to its complexity. *Hybrid Application Mapping* (HAM) approaches from the embedded domain address this challenge by combining extensive design-time analysis with lightweight runtime management [49, 65]. This dual approach enables rapid adaptation at runtime while benefiting from prior exploration.

#### 3.2.1 *Design-time Exploration.*

In HAM, design-time exploration generates a set of *operating points* through Design Space Exploration (DSE). Each operating point represents an application configuration and mapping, annotated with *non-functional characteristics* such as execution time or energy consumption. These operating points serve as the central link between design-time analysis and runtime resource management.

The goal of DSE is to identify *Pareto-optimal* operating points — those that are superior in at least one objective, such as lower execution time, reduced energy consumption, or fewer required cores (cf. the green rings in Figure 1). Various methods, from evolutionary algorithms [4, 38, 51, 72] to heuristics [33, 41, 45, 64], are used to find these points. Some approaches also distinguish different application scenarios [51, 58, 60, 68]. Non-functional characteristics can be obtained through models, traces, and static analysis [11, 23, 38], or by direct measurement.

#### 3.2.2 *Runtime Resource Management.*

At runtime, the RM uses the provided Pareto-optimal operating points to select one per application, aiming to maximize efficiency while respecting platform constraints. This optimization can be formalized as follows:

$$\textbf{minimize} \sum_{\sigma\in\Sigma} o^{\sigma}_{x_\sigma}[\zeta], \tag{1a}$$

$$\textbf{subject to} \sum_{\sigma\in\Sigma} o^{\sigma}_{x_\sigma}[\vec{r}] \leq \vec{R}. \tag{1b}$$

Here, $x_\sigma$ denotes the operating point selected for job $\sigma \in \Sigma$. Each operating point $o^{\sigma}_{i} \in O^{\sigma}$ is annotated with an *energy–utility cost* $o^{\sigma}_{i}[\zeta]$ and a *resource vector* $o^{\sigma}_{i}[\vec{r}]$, which specifies the required number of cores of each type. The vector $\vec{R}$ represents the platform's total resource capacity. Equation (1a) minimizes the system-wide energy-utility cost, while Equation (1b) ensures that resource demand does not exceed availability.

A variety of approaches have been proposed to solve this optimization problem. Some methods select operating points iteratively [64, 72], while others solve it jointly, e.g., using ILP solvers [9]. A common joint approach is to treat this problem as a *Multiple choice Multi-dimensional Knapsack Problem* (MMKP) [39], where the goal is to select one item per group with multidimensional weights while maximizing overall value. Since MMKP is NP-hard [50], computing an exact solution within reasonable time — especially for a large number of applications and operating points — is computationally challenging. Consequently, MMKP-based methods rely on approximate solutions, employing greedy heuristics [76], Pareto algebra principles [62], or Lagrangian relaxation [68, 73, 74].

While HAM benefits from extensive offline analyses to guide runtime decisions, its reliance on design-time exploration limits applicability in general-purpose systems, where applications may be unknown. AdaMD [6] addresses this limitation by using hardware performance counters to analyze application behavior at runtime. Although effective, these counters may not fully capture actual application performance.

## 3.3 Controlling In-Application Adaptivity

To fully leverage heterogeneous processors, applications should expose adaptivity features. Some of these are intrinsic to the application itself. An example is dynamic load distribution across heterogeneous cores. Even workload distribution can lead to imbalances, as tasks on energy-efficient cores take longer to complete. To address this, solutions for dynamic workload distribution have been proposed for application models like process networks [31, 32] and OpenMP [54].

Other forms of adaptivity require external control through *adaptivity knobs*, allowing runtime systems to dynamically adjust an application configuration, e.g., application topology, parallelization degree, or even algorithms [32, 59, 70]. For example, AdaPNet [59] selects configurations at runtime based on resource availability and historical data. However, its applicability is limited to process networks. Despite these advances, no existing resource-management system provides broad support for in-application adaptivity across diverse models.

## 3.4 Comparison with HARP

The approaches above address parts of the resource-management challenge, but none combines all aspects of them into a single methodology. HARP integrates these perspectives as follows.

First, unlike traditional OS schedulers (Section 3.1), HARP makes applications explicitly aware of allocation decisions. It introduces a two-way communication channel between applications and the RM, enabling *collaboration* for greater efficiency: applications adapt to allocations while providing feedback that guides future decisions.

Second, similar to HAM (Section 3.2), HARP leverages operating points supplied through application descriptions (e.g., from DSE). Unlike HAM, however, it does not depend on design-time exploration: operating points can also be generated at runtime, supporting previously unseen applications. AdaMD [6] also avoids reliance on design-time analysis, but is limited to performance-counter data. In contrast, HARP can also incorporate application-specific metrics that reflect *true* utility via its communication channel.

Finally, HARP extends resource management beyond core placement. Whereas prior works often target specific application models (Section 3.3), HARP generalizes application support through a flexible framework. Its `libharp` library offers built-in support for common application models and an interface for integrating new ones, allowing applications to tune internal parameters such as parallelism or other adaptivity knobs.

In summary, no existing system addresses all these aspects jointly. By combining system-level coordination, runtime exploration, and application-level adaptivity, HARP provides a comprehensive resource-management framework that advances the state of the art.

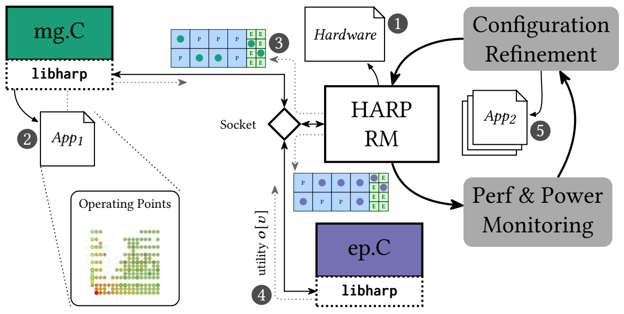


**Figure 2: Overview of the HARP management approach and system design with two example applications.**

## 4 HARP Design

Building on the limitations of prior approaches, we now present the design of *Heterogeneity-aware Adaptive Resource Partitioning (HARP)*, our resource-management framework. Figure 2 illustrates its design. HARP consists of two main components: the HARP Resource Manager (HARP RM) and the `libharp` library[1]. A single instance of the HARP RM oversees all managed applications, while each application runs its own `libharp` instance. The RM makes high-level decisions on resource allocations and core affinities, working alongside the OS scheduler (see Figure 4 in §4.3), which handles low-level task scheduling. The `libharp` library facilitates two-way communication with the RM, ensuring coordination across diverse application models.

HARP RM updates resource allocations whenever an application starts, exits, or other system events occur. These decisions are guided by the hardware description (❶) and by the *operating points* (§4.1.2). Operating points may be parsed from application description files (❷) via `libharp` or generated at runtime through HARP's fast exploration heuristics (❺), detailed in Section 5. For each application, the RM selects one operating point from the available set, adjusts it to ensure spatial isolation among running applications, and communicates the decision through `libharp` (❸), which performs application-specific adaptations and optionally returns performance feedback (❹).

This split design, where the central RM makes global allocation decisions, each application adapts via its own `libharp` instance, and both communicate through a two-way channel, provides high flexibility and supports a wide range of use cases. Furthermore, hardware characteristics are not hard-coded in the HARP RM but supplied dynamically at runtime via the hardware description file, allowing system administrators to fine-tune HARP's behavior. Overall, HARP's design simplifies resource management while integrating well across applications and scenarios.

## 4.1 Application Support via `libharp`

`libharp` serves as an intermediary between the RM and the applications, handling application registration, dynamically adjusting application configurations based on RM decisions, and providing utility metrics. It ensures effective interaction and runtime adaptations using standardized communication protocols. While all the communication is handled by one interface with the RM, `libharp`

[1]Both components are available open-source at: https://github.com/TUD-OS/harp

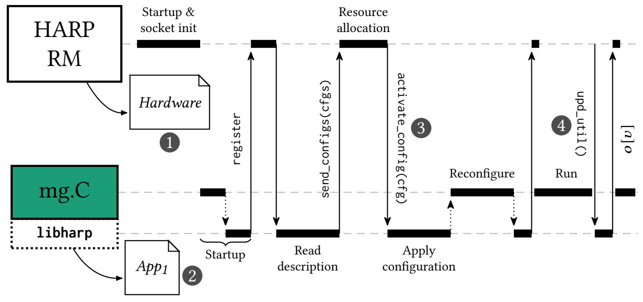


**Figure 3: Typical control flow between a managed application and the HARP resource manager.**

can manage applications differently based on their supported allocation type (§4.1.2) and adaptivity (§4.1.3).

*4.1.1 Communication Protocol.* The library communicates with the RM through a simple interface that is handled using `protobuf` messages over Unix sockets. A typical control flow is shown in Figure 3, outlining the interaction between the managed application `mg.C` and the HARP RM.

**(1) Registration Request:** Upon startup, `libharp` initializes the connection with the RM by sending a registration request through the RM's Unix socket. This request includes the application's PID and the supported resource type (§4.1.2). At this point, `libharp` also creates a dedicated Unix socket for receiving push messages from the RM.

**(2) Operating Points and Utility Subscription:** After registration, the application provides the RM with a list of operating points from the application description file (❷), if these are available. Furthermore, the library might indicate further functionality to the RM depending on the application adaptivity type (§4.1.3), for example that it can provide real-time utility metrics.

**(3) Operating Point Activation:** After finishing the first initial setup steps, the RM generates a new resource allocation. It communicates this decision to `libharp`, specifying the selected operating point and its allocation on the processing resources (❸). The library will accordingly update the application configuration to the provided resources. The concrete changes to the application depend on the application type and its available configuration knobs. Such reconfiguration messages from the RM not only happen at the startup of the application but can also occur whenever the RM is triggered, e.g. when other applications start.

**(4) Utility Feedback:** If the utility feedback feature is indicated, the RM periodically requests the current utility from `libharp` (❹) to refine operating point tables (❺).

*4.1.2 Operating Points and Allocation Granularity.* Operating points are the primary data structures linking the HARP RM and `libharp`. For each application, HARP maintains a set of operating points, each representing a distinct configuration variant. An operating point encodes (1) an in-application configuration, (2) a resource allocation, and (3) associated non-functional characteristics.

The first two components — application configuration and resource allocation — together define the overall configuration variant. In HARP, these can be expressed at two levels of granularity: fine-grained and coarse-grained.

**Fine-grained operating points** specify detailed mappings of individual application threads to processor cores and include in-application parameter values controlled through *adaptivity knobs.*

**Coarse-grained operating points**, in contrast, use a compact representation based on an *extended resource vector* that defines the resource requirements, indicating the number of cores and their hardware thread usage per core type. For example, if an application uses 4 E-cores and 3 P-cores (with simultaneous multithreading), where two P-cores use two hardware threads and the third only one, the extended resource vector is $[1, 2, 4]^\top$. This vector applies to the entire application, implying a uniform thread distribution. In this case, only adaptivity knobs derivable from the resource vector, such as the parallelization degree, can be managed by HARP.

The distinction between coarse- and fine-grained operating points stems from their differing strengths and trade-offs. Coarse-grained points are simpler since only core combinations have to be distinguished, offering a uniform structure across applications. This simplicity allows the HARP RM to explore operating points at runtime (Section 5), whereas fine-grained points enable richer application structures and more adaptivity knobs but lead to a much larger, more complex search space, making online exploration impractical.

Notably, even in the case of fine-grained operating points, the RM does not receive detailed thread-to-core mappings or adaptivity knob values. Instead, `libharp` communicates only the extended resource vector, just as with coarse-grained allocations, along with the associated non-functional characteristics (see §4.2.1).

*4.1.3 Application Adaptivity Types.* In HARP, applications are classified based on their adaptivity options: static, scalable, or custom.

**Static applications** are applications without any runtime adaptation mechanisms. Both allocation types can be combined with such applications, for example with threads mapped to specific cores (fine-grained), or by just restricting the entire application to a subset of the cores (coarse-grained). `libharp` natively supports static applications: registration with the RM is done automatically upon library load and the application threads are managed via intercepting `pthread_*()` functions. A drawback, however, is that when a static application runs more threads than allocated cores, performance may degrade as the OS scheduler will time-multiplex the assigned cores to the threads.

**Scalable applications**, on the other side, come with some type of runtime scaling. Libraries like OpenMP and Intel TBB enable implicit data-level parallelism, classifying these as *scalable* applications. Since parallel worker threads in these applications usually perform the same operations on different data, they naturally support coarse-grained resource allocations. Typically, the parallelization degree is set at the start, making these applications *moldable* [18]. `libharp` extends their capabilities by making them *malleable*, i.e., allowing the parallelization degree to be adjusted dynamically at runtime via a built-in adaptivity knob. For Intel TBB and OpenMP, this knob is implemented by hooking into library-internal functions. For example, in OpenMP, we hook in the `GOMP_parallel` function and adjust the `num_threads` variable to the maximum of the user-given number and the parallelization degree provided by the HARP RM. This adaptation matches the number of worker threads to hardware threads according to the corresponding extended resource vector, which helps to prevent time-sharing processor cores, which the static applications may suffer from. However, `libharp` can also treat applications as scalable when they provide their own scalability adaptivity knob. For instance, we developed a wrapper library for TensorFlow [17] that demonstrates this capability.

**Custom applications** extend `libharp` to support application-specific adaptations. These *custom* applications typically use fine-grained operating points. For example, we developed a `libharp` extension for Kahn Process Networks (KPN) [31] that scales specific *parallel regions* within the application, allowing for fine-grained resource allocation across different phases [32]. Other extensions might leverage the communicated resource allocations by selecting alternative algorithms, using specialized code paths, or handling ISA-extension differences between core types.

#### 4.1.4 Making Applications HARP-ready.

Most static and scalable applications are supported by `libharp` without further modifications. The library automatically detects supported runtime libraries and application threads, adapting them through function hooks.

For custom adaptations, developers only need to register callbacks via a `libharp` interface. These callbacks use the selected operating point and resources to reconfigure the application, for example by adjusting internal parallelism or other adaptivity knobs. `libharp` invokes the callbacks whenever the HARP RM assigns a new allocation. Utility metric feedback can be integrated in the same way via a dedicated interface.

Overall, making an application HARP-ready is straightforward. The interfaces are simple and require minimal developer effort, while `libharp` and HARP RM handle the rest. As shown in Section 6, this level of integration already yields significant improvements in energy efficiency.

## 4.2 Resource Allocation

The HARP RM allocates heterogeneous resources to concurrently running applications, balancing the resource needs while optimizing system energy efficiency.

#### 4.2.1 Non-Functional Characteristics.

Instead of using execution time and energy consumption, HARP relies on *instant* metrics: utility and power consumption, annotated in the operating points.

The *power consumption* metric $o[p]$ represents the power attributed to the application. The *utility* metric $o[v]$ measures the useful work performed by the application, either via generic "Instructions Per Second" (IPS), obtained from `perf`, or application-specific metrics like transactions per second, reflecting the meaningful work more accurately. Utility data may come from description files or from a running application. To address varying interpretations and magnitudes of utility across applications, HARP normalizes the utility value by dividing it by the maximum utility observed for that application, denoted as $o[v^*]$.

#### 4.2.2 Energy-Efficient Resource Allocation.

HARP applies the state-of-the-art resource allocation techniques discussed in §3.2.2, defining the energy-utility cost for each operating point to guide optimization. The *energy-utility cost* $o[\zeta]$ is adapted from the traditional Energy-Delay Product (EDP) formula balancing energy efficiency with application performance [40, 46]. Assuming utility is inversely proportional to delay, the energy-utility cost is calculated as:

$$o[\zeta] = \left(\frac{o[p]}{o[v^*]}\right) \cdot \left(\frac{1}{o[v^*]}\right) \tag{2}$$

These costs are used in the optimization problem outlined in Equation (1). HARP employs a state-of-the-art approximation algorithm based on Lagrangian relaxation, which solves the problem under relaxed constraints, selects the final operating points that meet the resource constraints, and finally, finds a concrete allocation of resources to applications, ensuring no overlap. Our implementation is similar to the one by Wildermann et al. [73, 74].

*Limitations.* The resource allocation algorithm may not find a suitable operating point for some of the applications due to prior allocations. This situation may occur if the number of managed applications exceeds the number of available resources. In such cases, HARP temporarily relaxes the constraint in Equation (1b), allowing applications to execute in co-allocation. Since co-allocation adversely affects the performance, HARP does not use performance monitoring, as discussed in Section 5.1.

## 4.3 HARP within the System Stack

We envision HARP not as a replacement, but as an extension of the existing system stack. Instead of duplicating functionality, HARP builds on existing kernel subsystems — like the Linux scheduler and `perf` framework — to collect runtime data like hardware counters and the system energy. Figure 4 shows how HARP, `libharp`, and the existing OS environment interact. In this setup, HARP acts as a central user-space resource manager, similar to system services like `systemd`, `OpenRC`, or Apple's `launchd`. Unlike existing user-level schedulers [26, 30, 75, 77], however, HARP does not influence scheduling decisions directly. Instead, it focuses on efficiently assigning resources to applications.

All configuration data — including the hardware information and the application-specific operating points — is stored in a directory such as `/etc/harp`. To reduce user burden, we propose a deployment model where the hardware description is provided by the vendor or auto-generated during setup, while application profiles are bundled with the applications. If no profile is available at launch, HARP can dynamically generate initial operating points based on observed behavior (see Section 5). These profiles are refined over time, enabling self-improving resource management. Centralizing

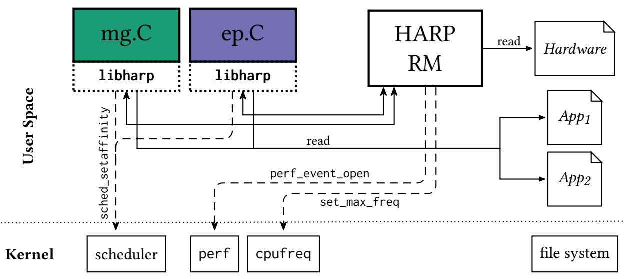


**Figure 4: Overview about the interactions of HARP and other existing operating system components.**

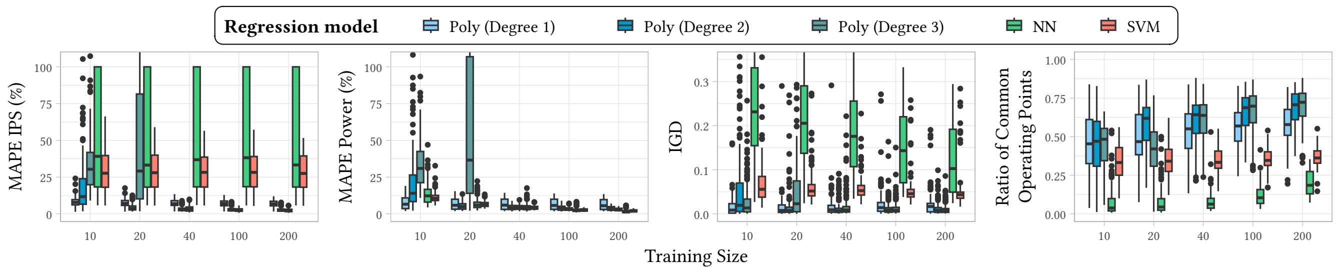


**Figure 5: Evaluation of several regression models in terms of Mean Absolute Percentage Error (MAPE) for IPS and Power (lower is better), Inverted Generational Distance (IGD) (lower is better), and the ratio of common operating points in the Pareto-front (higher is better) across 15 different applications on the Intel Raptor Lake Core i9-13900K.**

configurations in a user-accessible directory also allows administrators and power users to inspect and adjust profiles, tailoring HARP's decisions to specific systems or workloads.

The variant of HARP used in our evaluation requires applications to actively register with the RM in order to be fully managed. Consequently, background tasks such as system daemons are not handled by HARP and thus time-share the processor with the managed applications under the regular Linux scheduler. However, in a production system, we envision HARP also integrating the background tasks into its global resource allocation strategy.

## 5 Runtime Exploration of Operating Points

Applications running on desktops and servers usually lack predefined application descriptions with operating points, and even if points are available, they may be imprecise due to hardware variations. To address this problem, HARP incorporates runtime exploration of operating points, a synergy between Design Space Exploration and Runtime Resource Management. This integration poses several challenges.

First, during exploration, we need to make precise online measurements to determine the utility and power consumption of the running applications. If applications do not provide their own utility metrics, Instructions Per Second (IPS) serves as a generic utility measure. However, both utility and power metrics fluctuate due to measurement noise and varying application stages, requiring periodic re-evaluation for a higher robustness and reliability of the resource allocation.

Second, exploring all possible operating points on a system at runtime is impractical. Thus, the HARP resource allocation algorithm not only relies on operating points with actually measured characteristics, but also approximates them for not yet explored operating points.

Third, the search for near-optimal configurations must be seamlessly integrated within the HARP RM. This integration ensures that operating point exploration does not compete with other concurrently running applications on processor cores. It also requires that the RM allocates sufficient resources to new applications, allowing them room for exploration without significantly undermining the performance of existing applications. To achieve this, we have enhanced the HARP RM to include the exploration of operating points directly during application execution. This process involves continuous performance monitoring and employing a regression model to adjust operating points, ensuring robust, Pareto-optimal configurations.

### 5.1 Performance and Power Monitoring

Accurate runtime performance monitoring is essential for refining Pareto fronts. If applications signal that they can provide their own utility metric, HARP RM bases its algorithm on these values. Otherwise, HARP RM relies on the Linux `perf` subsystem to track IPS. With proper configuration, `perf` multiplexes these measurements across applications automatically.

For power consumption, we rely on built-in power sensors such as the `RAPL` (Running Average Power Limit) counters on Intel machines. Since these typically measure system-wide energy, we build atop EnergAt [25], which monitors `RAPL` counters alongside thread execution metrics to estimate per-application power use.

EnergAt does not consider different core types in heterogeneous CPUs. To address this, we introduce power coefficients ($P^P = \gamma \cdot P^E$, determined offline) to attribute total energy consumption ($E_\Delta^{CPU}$) to P-cores ($E_\Delta^P$) and E-cores ($E_\Delta^E$):

$$E_\Delta^{CPU} = E_\Delta^P + E_\Delta^E = T_{total}^P \cdot P^P + T_{total}^E \cdot P^E \quad (3)$$

where $T_{total}^{P|E}$ represents the total CPU time on P-cores and E-cores, respectively. After approximating $E_\Delta^P$ and $E_\Delta^E$, we use the algorithm detailed by EnergAt to further attribute energy to applications running on homogeneous subsets of cores.

We validated this extension experimentally by comparing the attributed dynamic energy in multi-application scenarios (Section 6) with the same applications executed in isolation under identical configurations. The overall *Mean Absolute Percentage Error* (MAPE) was 8.76 %, confirming that the attribution is sufficiently accurate.

To smooth the inherent variability in measured utility and power, we apply an exponential moving average (EMA) with a smoothing factor of 0.1. This stabilizes short-term fluctuations while adapting to significant shifts in application behavior, ensuring accurate and responsive performance and power profiling.

### 5.2 Selection of the Regression Model

As regression models, we consider Polynomial Regression (degrees 1 to 3), Neural Networks (NN), and Support Vector Machines (SVM) for approximating utility and power consumption for yet unmeasured operating points. The models work with coarse-grained resource allocations, represented by the extended resource vector

(§4.1.2), which is used as input for predicting metrics. We used pre-measured data from 15 applications on the Intel Raptor Lake Core i9-13900K, training each model on subsets of different sizes with 10 random seeds for robustness.

As shown in the left two plots of Figure 5, Polynomial Regression improved in accuracy for both utility and power with larger training sets. Higher-degree models achieved greater accuracy at larger training sizes, albeit requiring more data points to converge. Conversely, NN and SVM models performed better in power predictions at larger training sizes but struggled with utility (IPS).

The right two plots show the comparison between the predicted and reference Pareto fronts (derived from the measured configurations) using Inverted Generational Distance (IGD) [13] and the ratio of common operating points. Polynomial models consistently outperformed SVM and NN in aligning with the reference Pareto front. Among the polynomial regression models, degrees 2 and 3 achieved better alignment with the reference front compared to the first-degree model. While both second- and third-degree models produced similar accuracy, the second-degree model was more efficient, requiring only 20 training points to converge. Based on these results, HARP currently uses a second-degree polynomial regression model for runtime exploration.

### 5.3 Runtime Exploration Algorithm

The runtime exploration of operating points must integrate seamlessly with the resource allocation algorithm, ensuring that operating points selected during exploration do not overlap with cores used by other applications. At the same time, sufficient resources must be allocated for efficient exploration without adversely affecting the performance of other applications.

We categorize the maturity of application operating points into three stages: (1) *Initial* stage, where there are insufficient measured operating points making approximations unreliable; (2) *Refinement* stage, with an intermediate number of measured points but still limited accuracy; and (3) *Stable* stage, where sufficient points have been explored for reliable approximations.

The integrated algorithm consists of two stages: it first performs multi-application resource allocation (Section 4.2), and then explores operating points within the bounds of the resources assigned to each application. Applications in the stable stage simply execute on the designated cores, while those in the initial and refinement stages continue exploration. Any unassigned cores are distributed evenly among such applications, allowing them to explore a broader configuration space. The specific exploration heuristic applied depends on the application's maturity stage.

In the *initial* stage, since there are insufficient measured points to create even a preliminary model, the goal is to distribute the measurements evenly across the configuration space to quickly develop a preliminary regression model. To achieve this, the heuristic selects the next operating point furthest from any previously measured configurations, based on the extended resource vector, to maximize diversity.

In the *refinement* stage, a preliminary regression model based on the earlier measurements becomes available, but it may still be imprecise and prone to anomalies, such as negative utility and power predictions. The purpose of this stage is to refine the model's accuracy by strategically selecting configurations for measurements.

The heuristic first prioritizes configurations with negative utility or power predictions, selecting the one with the largest combined error, defined as the geometric mean of their negative deviations (with positive values counted as zero).

If no configurations have negative predictions, the heuristic compares the primary regression model with an auxiliary one anchored by a "zero" point (zero utility and zero power for a configuration with no cores). The next operating point is then chosen based on the configuration with the largest combined discrepancy between the two models, measured as the geometric mean of their differences in predicted utility and power.

Each selected operating point undergoes a predefined number of measurements (in the evaluation, 20 measurements at 50 ms intervals). When the measurements are complete, a different operating point is selected for evaluation. This iterative process repeats until the application explores 25 different configurations and switches into the *stable* stage. Once in this stage, the resource allocation algorithm is invoked with a longer interval (every 100 measurements) to reassess the current allocation.

## 6 Evaluation

To demonstrate that HARP enhances the management of energy-aware tasks on heterogeneous processors, we evaluated it on two systems with distinct characteristics to highlight that our approach is generic and works across various types of heterogeneous CPUs. All experiments are fully reproducible using the accompanying artifact [67].

### 6.1 Evaluation Setup

As representative from the embedded computing domain, we evaluate on the Odroid XU3-E board [22], featuring a Samsung Exynos 5422 processor, which implements an Arm big.LITTLE architecture with two core islands: a four-core A15 (big) island and a four-core A7 (LITTLE) island. The system is equipped with 2 GB of memory and energy sensors for core islands, memory, and GPU. We run a custom-compiled Linux 6.6 kernel on the board, with full support for the Linux Energy-Aware-Scheduler (EAS) [15, 47].

From the desktop domain, we use an Intel Raptor Lake Core i9-13900K, one of Intel's latest heterogeneous processors. It consists of 8 high-performance P-cores (with SMT), and 16 energy-efficient E-cores (without SMT) and is equipped with 128 GB of memory. Energy measurements are captured using RAPL counters, which have been proven accurate in fine-grained energy measurements [14, 21, 27, 61, 66]. We run a custom-compiled Linux 6.4 kernel, based on the default Debian Testing kernel version, extended with a patch-set adding preliminary support for the Intel Thread Director (ITD) [44]. We further extended the patch-set to make the ITD classification of threads and the reference IPC per class determined by the hardware available to user-space. Inspired by Saez et al. [56] we implemented a version of HARP that uses ITD classifications to allocate processor cores to application threads.

For both platforms, we use the default frequency scaling governor (`powersave` on the Intel system and `schedutil` on the Odroid system, respectively) but limit the maximum frequency to prevent

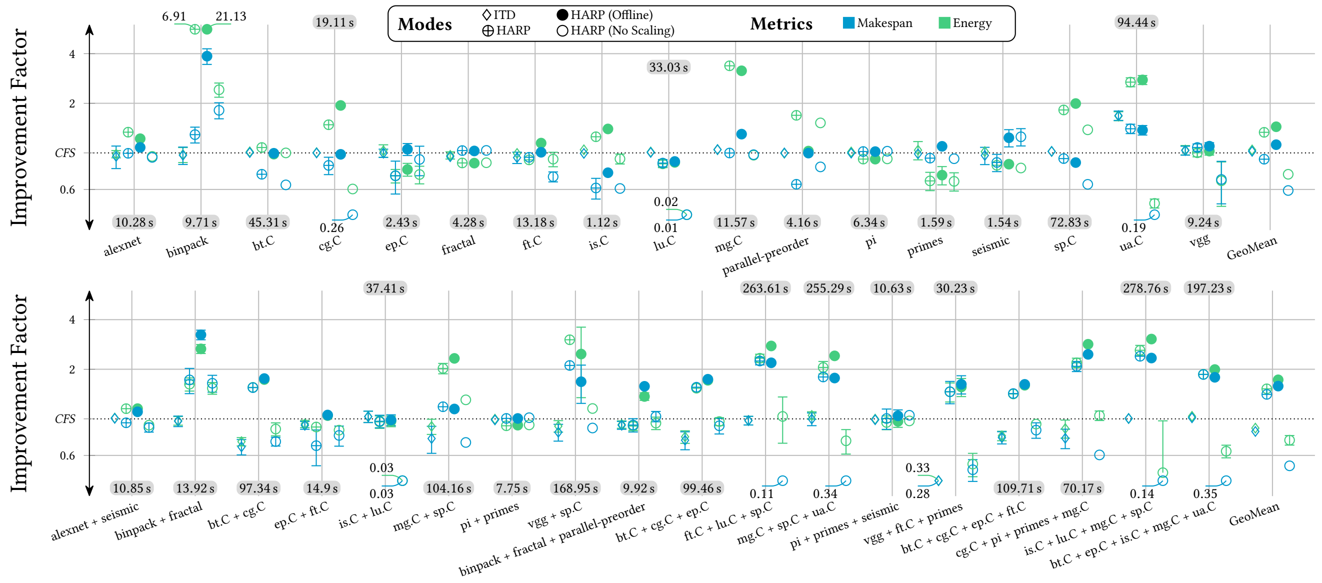


**Figure 6: Relative improvement factor of HARP and Intel Thread Director (ITD) over CFS on the Intel Raptor Lake Core i9-13900K (higher y-value is better). Results also include HARP with offline-generated operating points and HARP without application adaptation. Gray boxes indicate the average makespan for each scenario when executed with CFS.**

thermal throttling. This ensures that all cores can operate at their maximum allowed frequency throughout the benchmark execution: 4.6 GHz for the P-cores and 3.8 GHz for the E-cores on the Intel, and 1.2 GHz for LITTLE and 1.8 GHz for big cores on the Odroid.

## 6.2 Benchmarks

We use different sets of applications to evaluate the features of HARP. To test dynamic adaptability, we use the OpenMP implementations of the NAS Parallel Benchmarks [5], version 3.4.2 on both of the evaluated platforms. Since the Intel platform is more powerful than the Odroid board, we use the bigger class *C* of the benchmarks on Intel and the smaller class *A* on the Odroid.

On Intel, we also evaluate a selection of Intel Thread Building Blocks (Intel TBB) [48] benchmarks and two TensorFlow [17] applications. From the official Intel TBB repository we chose the benchmarks `binpack`, `fractal`, `parallel-preorder`, `pi`, `primes`, and `seismic`, as they cover a wide spectrum of the building blocks of the framework. TensorFlow is an open-source framework for machine-learning algorithms that is widely used. We implemented a HARP-enabled version of TensorFlow Lite [1] that can dynamically scale its parallelism at runtime. We evaluate two image recognition models, VGG [63] and AlexNet [35].

Additionally, we use two embedded KPN applications to test the custom extensions of HARP: `mandelbrot` for calculating the Mandelbrot set [32], and `lms` implementing Leighton-Micali Signatures [42]. Both applications are used in two versions: one with a static application topology (annotated with `static`) and another with implicit data-parallelism in KPNs [32], used to demonstrate dynamic adaptation. Since KPN applications are targeted to embedded platforms, we chose to evaluate them only on the Odroid system.

## 6.3 Intel Raptor Lake

We evaluate the performance and energy consumption of the benchmarks on the Intel Raptor Lake system, considering both single and multi-application scenarios. Each scenario is executed with HARP and compared against two baselines, namely, (1) *CFS*, which uses the built-in work-distribution techniques of Linux, and (2) *ITD* which uses our extended ITD-based allocator described earlier.

HARP dynamically generates operating points through online monitoring, while the ITD-based allocation relies on the hardware classification data. When comparing with the baseline, we show the performance of HARP with *stable* operating points, while its behavior during the learning process is discussed separately in Section 6.5. To demonstrate the potential of running offline design space exploration, we evaluate HARP with pre-generated operating points (*HARP (Offline)* in the figures). Finally, to show the importance of runtime adaptation in heterogeneous systems, we include results of HARP without application adaptation (*HARP (No Scaling)*).

For each scenario, we measure overall execution time (makespan) and total energy consumption, reporting average results from ten repetitions. Error bars are shown for relative errors exceeding 5 %. Figure 6 shows the results for all benchmarks, with geometric means provided for both single and multi-application scenarios. The results are reported as improvement factors over the baseline (*CFS*); an improvement factor of $F\times$ indicates that the resource manager executed $F\times$ faster or with that much less energy. Naturally, the higher $F$, the better.

### 6.3.1 *Single Application.*

In single-application scenarios, the ITD-based allocator shows results close to the baseline, with average improvement factors of 1.02× for execution time and 1.04× for energy consumption, both within the margin of error.

By contrast, HARP shows a stronger trade-off: it typically reduces energy consumption at the cost of slightly longer execution

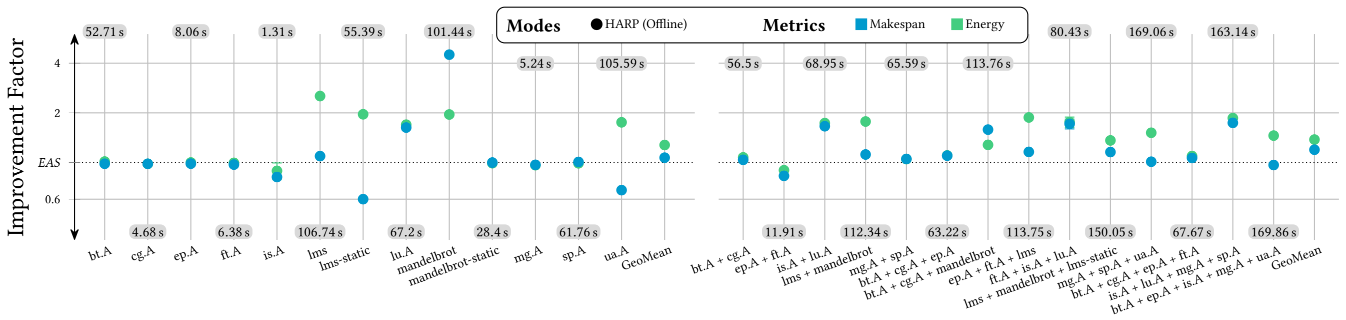


**Figure 7: Relative improvement factor of HARP over EAS on the Odroid XU3-E (higher y-value is better). Gray boxes show the average makespan of each scenario when executed with EAS.**

times. Given its cost function, which balances performance and energy, HARP often prioritizes one over the other depending on the application (e.g., `is`, `pi`, `seismic`). For `primes`, however, performance improved only slightly, while energy consumption degraded more significantly, even with HARP (Offline). As a short-running application, `primes` is affected by initial communication overhead, during which HARP interferes with the execution of `primes`.

A notable outlier is `binpack`, which achieves a 6.91× speedup and a 1.29× energy reduction. In the baseline execution, 32 threads contend on the same input queue throughout the whole execution, while HARP scales the application down to avoid this bottleneck and achieve much higher efficiency. In contrast, `lu` performs worse in both metrics. HARP selects a configuration with 21.5 % higher IPS than the one used by CFS, but execution time on this selected configuration is around 15 % longer. This illustrates that IPS, while reasonable, may not reflect true utility; with an application-specific utility metric, HARP would likely select a better configuration.

The `bt` benchmark shows that online exploration may miss the optimal point, whereas offline exploration selects a better one. The additional information from offline DSE provides HARP more complete behavior characteristics and leads to more optimal choices. This is particularly visible in the geometric means: HARP improves energy by 1.34× but reduces performance to 0.92×, while with offline-generated operating points it achieves both faster execution (1.22×) and lower energy consumption (1.44×).

Finally, HARP without application adaptation performs very poorly (geometric means of 0.60× for execution time and 0.74× for energy consumption), indicating the critical role of dynamic application adaptation for effective resource management.

#### 6.3.2 *Multiple Applications.*

The ITD-based resource allocation did not yield improvements in multi-application scenarios. Most scenarios achieved similar or worse performance than the baseline (except for `is + lu`), resulting in overall improvement factors of 0.84× for execution time and 0.88× for energy consumption.

By contrast, HARP delivers significant gains, with average improvement factors of 1.40× for execution time and 1.52× for energy consumption. As in the single-application scenarios, HARP with offline-generated operating points performs even better, reaching 1.58× speedup and 1.73× energy reduction.

Compared to single-application results, the improvements here are stronger, mainly because HARP scales applications down to match the available resources. Without such adaptation, HARP performs even worse than in the single-application case (0.52× for execution time and 0.74× for energy consumption).

#### 6.3.3 *Influences of Frequency Scaling.*

To assess the impact of DVFS on the HARP scheduling approach, we repeated all measurements using the `performance` governor instead of the default one. The `performance` governor configures the Linux frequency scaling driver to select higher frequencies more aggressively and disables various energy-efficiency features of the processor.

In contrast to our expectations, the change of the governor had only a minor effect. With the `performance` governor enabled, HARP achieves 1.44× lower energy consumption and 1.20× speedup across all measured scenarios. The corresponding values under the default governor were 1.42× and 1.14×, respectively.

Similar results were observed for the version of HARP that uses offline-generated operating points. Here, the measured improvement factors are 1.61× for energy consumption and 1.36× for execution time under the `performance` governor, compared to 1.58× and 1.34× with the default `powersave` governor.

These results suggest that the frequency governor of Linux has only a minor impact on HARP's overall performance and energy efficiency. Nevertheless, we believe that fully integrating the frequency selection in the resource allocation process could yield more improvements. We plan to explore this direction in future work (see Section 7).

## 6.4 Odroid XU3-E

In contrast to the Intel Raptor Lake, Odroid does not allow tracking performance counters simultaneously on the big and the LITTLE cluster. With no means for online monitoring, we only evaluate *HARP (Offline)*, i.e., based on offline pre-generated operating point tables. The baseline for this experiment is the Linux Energy-Aware Scheduler (*EAS*), leveraging a power model of the heterogeneous processor. Figure 7 shows the results of this experiment.

#### 6.4.1 *Single Application.*

For single-application scenarios on the Odroid XU3-E, HARP achieves a 1.07× speedup and a 1.27× reduction in energy consumption. The results for OpenMP benchmarks generally reflect those on Intel Raptor Lake, with HARP typically improving energy consumption at the cost of longer execution times (e.g., `ua`). The `is` benchmark shows an increase in both execution time and energy consumption, mainly due to its short runtime making the startup overhead of HARP more noticeable. In contrast, `lu`, a long-running benchmark, benefits from HARP.

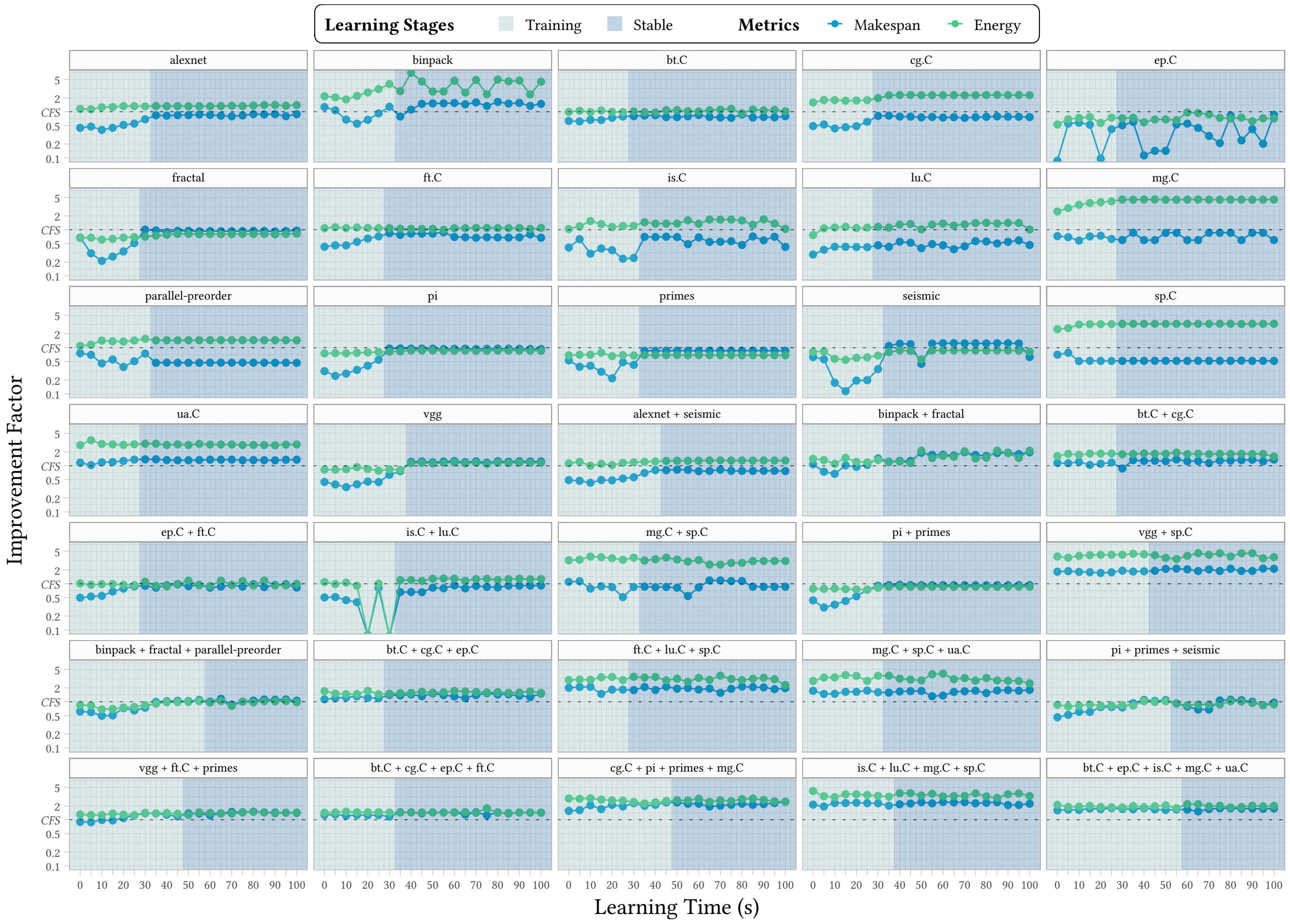


**Figure 8: Relative improvement factor of HARP over CFS during the learning phase on the Intel Raptor Lake Core i9-13900K (higher y-value is better).**

KPN applications, again, show improvements with HARP. HARP can successfully use the application knobs to tune the application to the available hardware resources, resulting in lower energy consumption for `lms` and improvements in both metrics for `mandelbrot`. `mandelbrot` in its static version behaves similarly to the baseline. The static version of `lms` behaves similar to `ua` and shows a lower energy consumption at the cost of a longer execution.

*6.4.2 Multiple Applications.* With multiple applications, HARP can improve resource usage, yielding average gains of 1.20× in makespan and 1.38× in energy consumption. Most scenarios benefit from HARP, with only `ep`+`ft` suffering in both metrics compared to the baseline. This is caused by thread movements between the big and LITTLE clusters during the application executions due to resource reassignments done by HARP.

## 6.5 Learning Operating Points

The previous results focused on HARP's performance after reaching the stable stage, where optimal operating points have already been learned. However, from a user's perspective, it is equally important to ensure that HARP performs acceptably even when no prior knowledge about application behavior is available. To evaluate this, we analyze HARP during the learning stage on Intel Raptor Lake.

Each test scenario starts with a warm-up phase, during which HARP monitors the running applications and learns their behavioral characteristics. To better understand this learning process, we snapshot the operating point tables at 5 s intervals.

Figure 8 illustrates the impact of these learning stages across all tested scenarios. Each dot in the figure shows the improvement factor for makespan and energy consumption for a corresponding snapshot. The background shading visualizes the learning status: the *stable* stage indicates that all applications have reached the stable stage (cf. Section 5.3), while the *learning* stage indicates that at least one application is still in the *initial* or *refinement* stages. As stated in Section 5.3, HARP continues refining operating points even after reaching the stable stage.

*6.5.1 Single-Application Scenarios.* In single-application scenarios, most applications exhibit fluctuating performance and energy efficiency during the training stages. Once stable operating points are reached, results becomes more consistent, though minor fluctuations may still occur. On average, the stable stages are achieved within 29.8 ± 5.9 s.

An exception is the ep application, which shows poor and unstable results even during the stable stage. This behavior is likely due to the very short execution time (2.43 s), which can lead to limited and potentially unreliable measurements for certain configurations. As the application is executed repeatedly, the exploration heuristic may still rely on these unreliable measurements when selecting subsequent operating points, introducing non-deterministic behavior into the learning process. This finding indicates the need to improve the exploration algorithm to ensure more robust and consistent learning outcomes.

*6.5.2 Multi-Application Scenarios.* In multi-application scenarios, a similar trend is observed: fluctuating metrics during the training stage, followed by stabilization once operating points become stable. Stable points are achieved on average within 36.6 ± 8.0 s, slightly longer than in single-application scenarios, reflecting limited resources available to each application, resulting in delays in the refinement stage.

Interestingly, as the number of applications increases, the difference between the results of learning and stable stages becomes smaller. In scenarios with four or five applications, the results observed during training closely match those observed in the stable stage. This is likely due to limited search spaces per application under resource constraints. During learning, HARP selects the configurations that use fewer resources, which are also preferred in the stable phase. As a result, outcomes of the training and stable stages become closely aligned.

## 6.6 Performance Overhead of HARP

As discussed in Section 2, resource management in a modern OS must be swift and lightweight to avoid negatively impacting the system performance. To assess the performance overhead of HARP with all its functionality enabled, we conducted additional measurements on the Raptor Lake system. We re-executed all application scenarios with HARP, including runtime exploration, configuration communication, and, if supported by applications, utility updates. However, the actual configuration assignment messages were ignored in `libharp`, leaving applications unadapted and thus be scheduled like the baseline CFS scheduler. This allowed us to isolate and measure the overhead of `perf` monitoring, energy estimation, the resource selection algorithm of the HARP RM, and all the communication between the RM and applications, separate from the performance gains from improved resource utilization. Overall, HARP introduced less than 1 % overhead when managing a single application, and 2.5 % in multi-application scenarios. Section 6.3, shows that this overhead is countered by HARP's superior resource allocation.

## 7 Conclusion

This paper introduces HARP, a resource management system designed for energy-aware applications on heterogeneous processors. HARP's key innovations include a simple interface between the application and a global resource manager for passing allocation decisions and an allocation algorithm that balances performance and energy consumption based on learned or pre-computed application characteristics. HARP supports a broad spectrum of application models, from static to scalable types, and even custom-specific ones with adaptivity features like reconfiguration options and algorithmic variations. Crucially, HARP does not require detailed knowledge of each application's adaptivity aspects. Instead, it manages only the essential information needed for efficient resource allocation (resource requirements and non-functional characteristics), with the rest handled on the application side.

Our evaluation on two heterogeneous systems shows that HARP can significantly improve application execution, both when given exclusive access to resources and when competing with concurrent applications. We reported improvements in terms of execution time and energy consumption of 12 % and 30 % for the Intel Raptor Lake Core i9-13900K and 12 % and 25 % for the Odroid XU3-E. For embedded KPN applications with custom adaptivity knobs, we demonstrated HARP's extensibility in providing fine-grained resource adaptations. Finally, we showed that HARP's overhead on running applications is small and usually countered by the gains it brings to the applications.

In summary, this work highlights the value of a simple yet effective interface to communicate allocation decisions between applications and the OS. Combining such an interface with application behavior characteristics in the form of resource requirements and non-functional characteristics, HARP considerably improves the system's overall energy-efficiency.

*Outlook.* HARP provides a rich framework, which allows future extensions. First, adding dynamic frequency-scaling control of the CPU would allow for even finer energy management. However this requires advanced behavior prediction techniques to handle the increased configuration complexity. Second, many applications exhibit distinct performance-energy characteristics across different execution stages. While the communication interface can be extended to allow applications to notify HARP of these stage transitions, a generic solution would require automatically detecting these stages without explicit application input. Finally, integrating techniques like Memguard [79], PALLOC [78], or GPUguard [19] would help to manage shared resource contention, thus further enhancing system performance.

## Acknowledgments

We would like to thank Andrés Goens and Marcus Hähnel for early discussions, as well as Dylan Gageot, Fabius Mayer-Uhma, and Marc Dietrich for their contribution in supporting Kahn Process Networks and TensorFlow applications. In addition, we thank the anonymous reviewers for their valuable input and feedback. The authors acknowledge the financial support by the Federal Ministry of Research, Technology and Space of Germany in the programme of "Souverän. Digital. Vernetzt." (joint project 6G-life, project number 16KISK001K) and the E4C project (16ME0426K). This work also received funding from the EU Horizon Europe Programme under grant agreement No 101135183 (MYRTUS). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

## References

[1] 2023. TensorFlow Lite. https://www.tensorflow.org/lite. [Online; accessed 18-October-2024].

[2] Murali Annavaram, Ed Grochowski, and John Shen. 2005. Mitigating Amdahl's law through EPI throttling. In *32nd International Symposium on Computer Architecture (ISCA'05)*. 298–309. doi:10.1109/ISCA.2005.36

[3] Apple. 2020. Apple unleashes M1. https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/. [Online; accessed 18-October-2024].

[4] Giuseppe Ascia, Vincenzo Catania, and Maurizio Palesi. 2004. Multi-Objective Mapping for Mesh-Based NoC Architectures. In *Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (Stockholm, Sweden) *(CODES+ISSS '04)*. Association for Computing Machinery, New York, NY, USA, 182–187. doi:10.1145/1016720.1016765

[5] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, and S.K. Weeratunga. 1991. The NAS Parallel Benchmarks. *Int. J. High Perform. Comput. Appl.* 5, 3 (sep 1991), 63–73. doi:10.1177/109434209100500306

[6] Karunakar R. Basireddy, Amit Kumar Singh, Bashir M. Al-Hashimi, and Geoff V. Merrett. 2020. AdaMD: Adaptive Mapping and DVFS for Energy-Efficient Heterogeneous Multicores. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2020), 2206–2217. doi:10.1109/TCAD.2019.2935065

[7] Michela Becchi and Patrick Crowley. 2006. Dynamic thread assignment on heterogeneous multiprocessor architectures. In *Proceedings of the 3rd Conference on Computing Frontiers* (Ischia, Italy) *(CF '06)*. Association for Computing Machinery, New York, NY, USA, 29–40. doi:10.1145/1128022.1128029

[8] Carlos Bilbao, Juan Carlos Saez, and Manuel Prieto-Matias. 2023. Flexible system software scheduling for asymmetric multicore systems with PMCSched: A case for Intel Alder Lake. *Concurrency and Computation: Practice and Experience* 35, 25 (2023), e7814. doi:10.1002/cpe.7814

[9] Enrico Bini, Giorgio Buttazzo, Johan Eker, Stefan Schorr, Raphael Guerra, Gerhard Fohler, Karl-Erik Arzen, Vanessa Romero, and Claudio Scordino. 2011. Resource Management on Multicore Systems: The ACTORS Approach. *IEEE Micro* 31, 3 (2011), 72–81. doi:10.1109/MM.2011.1

[10] Björn B. Brandenburg, John M. Calandrino, and James H. Anderson. 2008. On the Scalability of Real-Time Scheduling Algorithms on Multicore Platforms: A Case Study. In *2008 Real-Time Systems Symposium*. 157–169. doi:10.1109/RTSS.2008.23

[11] Jeronimo Castrillon, Andreas Tretter, Rainer Leupers, and Gerd Ascheid. 2012. Communication-aware mapping of KPN applications onto heterogeneous MPSoCs. In *Proceedings of the 49th Annual Design Automation Conference* (San Francisco, California) *(DAC '12)*. Association for Computing Machinery, New York, NY, USA, 1266–1271. doi:10.1145/2228360.2228597

[12] Tim Chen. 2023. Enable Cluster Scheduling for x86 Hybrid CPUs. https://lore.kernel.org/lkml/cover.1688770494.git.tim.c.chen@linux.intel.com/. [Online, accesses 18-October-2024].

[13] Carlos A. Coello Coello and Margarita Reyes Sierra. 2004. A Study of the Parallelization of a Coevolutionary Multi-objective Evolutionary Algorithm. In *MICAI 2004: Advances in Artificial Intelligence*, Raúl Monroy, Gustavo Arroyo-Figueroa, Luis Enrique Sucar, and Humberto Sossa (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 688–697.

[14] Maxime Colmant, Mascha Kurpicz, Pascal Felber, Loïc Huertas, Romain Rouvoy, and Anita Sobe. 2015. Process-Level Power Estimation in VM-based Systems. In *Proceedings of the Tenth European Conference on Computer Systems* (Bordeaux, France, 2015-04) *(EuroSys '15)*. ACM. doi:10.1145/2741948.2741971

[15] Linux Kernel Community. 2024. Energy Aware Scheduling. https://docs.kernel.org/scheduler/sched-energy.html. [Online; accessed 18-October-2024].

[16] Ian Cutress. 2021. Thread Director: Windows 11 Does It Best. https://www.anandtech.com/show/16959/intel-innovation-alder-lake-november-4th/3. [Online; accessed 18-October-2024].

[17] TensorFlow Developers. 2022. TensorFlow. *Zenodo* (2022).

[18] Dror G. Feitelson and Larry Rudolph. 1996. Toward Convergence in Job Schedulers for Parallel Supercomputers. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (IPPS '96)*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–26. doi:10.1007/BFb0022284

[19] Björn Forsberg, Andrea Marongiu, and Luca Benini. 2017. GPUguard: Towards supporting a predictable execution model for heterogeneous SoC. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 318–321. doi:10.23919/DATE.2017.7927008

[20] Peter Greenhalgh. 2011. big.LITTLE processing with arm cortex-a15 & cortex-a7. https://www.eetimes.com/big-little-processing-with-arm-cortex-a15-cortex-a7/. *ARM White paper* 17 (2011). [Online; accessed 18-October-2024].

[21] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. 2015. An Energy Efficiency Feature Survey of the Intel Haswell Processor. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. 896–904. doi:10.1109/IPDPSW.2015.70

[22] Marcus Hähnel and Hermann Härtig. 2014. Heterogeneity by the Numbers: A Study of the ODROID XU+E big.LITTLE Platform. In *6th Workshop on Power-Aware Computing and Systems (HotPower 14)*. USENIX Association, Broomfield, CO. https://www.usenix.org/conference/hotpower14/workshop-program/presentation/hahnel

[23] Marcus Hähnel and Till Smejkal. 2018. Modular Energy Modeling Using Energy/Utility. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering* (Berlin, Germany) *(ICPE '18)*. Association for Computing Machinery, New York, NY, USA, 73–78. doi:10.1145/3185768.3186311

[24] Christopher Harper. 2024. A third AMD Strix Point Ryzen AI CPU has been officially confirmed— Ryzen AI 9 HX 375 debuts just above HX 370. https://www.tomshardware.com/pc-components/cpus/a-third-amd-strix-point-ryzen-ai-cpu-has-been-officially-confirmed-the-ryzen-ai-9-hx-375-debuts-just-above-hx-370. [Online; accessed 18-October-2024].

[25] Hongyu Hè, Michal Friedman, and Theodoros Rekatsinas. 2023. EnergAt: Fine-Grained Energy Attribution for Multi-Tenancy. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems* (Boston, MA, USA) *(HotCarbon '23)*. Association for Computing Machinery, New York, NY, USA, Article 4, 8 pages. doi:10.1145/3604930.3605716

[26] Jack Tigar Humphries, Neel Natu, Ashwin Chaugule, Ofir Weisse, Barret Rhoden, Josh Don, Luigi Rizzo, Oleg Rombakh, Paul Turner, and Christos Kozyrakis. 2021. ghost: Fast & flexible user-space delegation of linux scheduling. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 588–604.

[27] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. 2012. Measuring Energy Consumption for Short Code Paths Using RAPL. (2012), 13–17. doi:10.1145/2425248.2425252

[28] Intel. 2022. Intel Unveils 12th Gen Intel Core, Launches World's Best Gaming Processor, i9-12900K. https://www.intel.com/content/www/us/en/newsroom/news/12th-gen-core-processors.html. [Online; accessed 18-October-2024].

[29] Intel Corporation. 2023. What Is Intel® Thread Director? https://www.intel.com/content/www/us/en/support/articles/000097053/processors/intel-core-processors.html [Online; accessed 18-October-2024].

[30] Yuekai Jia, Kaifu Tian, Yuyang You, Yu Chen, and Kang Chen. 2024. Skyloft: A General High-Efficient Scheduling Framework in User Space. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. 265–279.

[31] Gilles Kahn. 1974. The Semantics of a Simple Language for Parallel Programming. In *Information Processing 74: Proceedings of the IFIP Congress 74, Stockholm, Sweden, August 1974*, J.L. Rosenfeld (Ed.). North-Holland, Amsterdam, Netherlands, 471–475.

[32] Robert Khasanov, Andrés Goens, and Jeronimo Castrillon. 2018. Implicit Data-Parallelism in Kahn Process Networks: Bridging the MacQueen Gap. In *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms* (Manchester, United Kingdom) *(PARMA-DITAM '18)*. Association for Computing Machinery, New York, NY, USA, 20–25. doi:10.1145/3183767.3183790

[33] Robert Khasanov, Julian Robledo, Christian Menard, Andrés Goens, and Jeronimo Castrillon. 2021. Domain-specific hybrid mapping for energy-efficient baseband processing in wireless networks. *ACM Transactions on Embedded Computing Systems (TECS). Special issue of the International Conference on Compilers, Architecture, and Synthesis of Embedded Systems (CASES)* 20, 5s, Article 60 (2021), 26 pages. doi:10.1145/3476991

[34] David Koufaty, Dheeraj Reddy, and Scott Hahn. 2010. Bias scheduling in heterogeneous multi-core architectures. In *Proceedings of the 5th European Conference on Computer Systems* (Paris, France) *(EuroSys '10)*. Association for Computing Machinery, New York, NY, USA, 125–138. doi:10.1145/1755913.1755928

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2017. ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60, 6 (2017), 84–90.

[36] Rakesh Kumar, Dean M. Tullsen, Parthasarathy Ranganathan, Norman P. Jouppi, and Keith I. Farkas. 2004. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. *SIGARCH Comput. Archit. News* 32, 2 (March 2004), 64. doi:10.1145/1028176.1006707

[37] Zhiye Liu. 2023. AMD Phoenix 2 Review Evaluates Zen 4, Zen 4c Performance. https://www.tomshardware.com/news/amd-phoenix-2-review-evaluates-zen-4-zen-4c-performance. [Online; accessed 18-October-2024].

[38] Giovanni Mariani, Vlad-Mihai Sima, Gianluca Palermo, Vittorio Zaccaria, Cristina Silvano, and Koen Bertels. 2012. Using multi-objective design space exploration to enable run-time resource management for reconfigurable architectures. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1379–1384. doi:10.1109/DATE.2012.6176578

[39] Silvano Martello and Paolo Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA.

[40] Alain J Martin. 2001. Towards an energy complexity of computation. *Inform. Process. Lett.* 77, 2-4 (2001), 181–187.

[41] Giuseppe Massari, Edoardo Paone, Patrick Bellasi, Gianluca Palermo, Vittorio Zaccaria, William Fornaciari, and Cristina Silvano. 2014. Combining application adaptivity and system-wide Resource Management on multi-core platforms.

In *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*. 26–33. doi:10.1109/SAMOS.2014.6893191

[42] David McGrew, Michael Curcio, and Scott Fluhrer. 2019. Leighton-Micali Hash-Based Signatures. RFC 8554. doi:10.17487/RFC8554

[43] Jeffrey C. Mogul, Jayaram Mudigonda, Nathan Binkert, Parthasarathy Ranganathan, and Vanish Talwar. 2008. Using Asymmetric Single-ISA CMPs to Save Energy on Operating Systems. *IEEE Micro* 28, 3 (2008), 26–41. doi:10.1109/MM.2008.47

[44] Ricardo Neri. 2023. Introduce classes of tasks for load balance. https://lore.kernel.org/lkml/20230613042422.5344-1-ricardo.neri-calderon@linux.intel.com/. [Online, accessed 18-October-2024].

[45] Gereon Onnebrink, Ahmed Hallawa, Rainer Leupers, Gerd Ascheid, and Awaid-Ud-Din Shaheen. 2019. A Heuristic for Multi Objective Software Application Mappings on Heterogeneous MPSoCs. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference* (Tokyo, Japan) *(ASPDAC '19)*. Association for Computing Machinery, New York, NY, USA, 609–614. doi:10.1145/3287624.3287651

[46] Paul I Pénzes and Alain J. Martin. 2002. Energy-delay efficiency of VLSI computations. In *Proceedings of the 12th ACM Great Lakes Symposium on VLSI* (New York, New York, USA) *(GLSVLSI '02)*. Association for Computing Machinery, New York, NY, USA, 104–111. doi:10.1145/505306.505330

[47] Quentin Perret. 2019. Energy Aware Scheduling (EAS) in Linux 5.0. https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/energy-aware-scheduling-in-linux. [Online; accessed 18-October-2024].

[48] Chuck Pheatt. 2008. Intel® threading building blocks. *Journal of Computing Sciences in Colleges* 23, 4 (2008), 298–298.

[49] Behnaz Pourmohseni, Michael Glaß, Jörg Henkel, Heba Khdr, Martin Rapp, Valentina Richthammer, Tobias Schwarzer, Fedor Smirnov, Jan Spieck, Jürgen Teich, Andreas Weichslgartner, and Stefan Wildermann. 2020. Hybrid Application Mapping for Composable Many-Core Systems: Overview and Future Perspective. *Journal of Low Power Electronics and Applications* 10, 4 (2020). doi:10.3390/jlpea10040038

[50] Jakob Puchinger, Günther R. Raidl, and Ulrich Pferschy. 2010. The Multidimensional Knapsack Problem: Structure and Algorithms. *INFORMS J. on Computing* 22, 2 (apr 2010), 250–265. doi:10.1287/ijoc.1090.0344

[51] Wei Quan and Andy D. Pimentel. 2015. A Hybrid Task Mapping Algorithm for Heterogeneous MPSoCs. *ACM Trans. Embed. Comput. Syst.* 14, 1, Article 14 (jan 2015), 25 pages. doi:10.1145/2680542

[52] João Gabriel Reis and Antônio Augusto Fröhlich. 2017. OS Support for Adaptive Components in Self-aware Systems. *SIGOPS Oper. Syst. Rev.* 51, 1 (Sept. 2017), 101–112. doi:10.1145/3139645.3139663

[53] João Gabriel Reis and Antônio Augusto Fröhlich. 2016. Mutant Components: Efficiently Managing Multiple Implementations. In *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*. IEEE, 101–108. doi:10.1109/SBESC.2016.023

[54] Juan Carlos Saez, Fernando Castro, and Manuel Prieto-Matias. 2020. Enabling performance portability of data-parallel OpenMP applications on asymmetric multicore processors. In *Proceedings of the 49th International Conference on Parallel Processing* (Edmonton, AB, Canada) *(ICPP '20)*. Association for Computing Machinery, New York, NY, USA, Article 51, 11 pages. doi:10.1145/3404397.3404441

[55] Juan Carlos Saez, Alexandra Fedorova, Manuel Prieto, and Hugo Vegas. 2010. Operating system support for mitigating software scalability bottlenecks on asymmetric multicore processors. In *Proceedings of the 7th ACM International Conference on Computing Frontiers* (Bertinoro, Italy) *(CF '10)*. Association for Computing Machinery, New York, NY, USA, 31–40. doi:10.1145/1787275.1787281

[56] Juan Carlos Saez and Manuel Prieto-Matias. 2022. Evaluation of the Intel thread director technology on an Alder Lake processor. In *Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems* (Virtual Event, Singapore) *(APSys '22)*. Association for Computing Machinery, New York, NY, USA, 61–67. doi:10.1145/3546591.3547532

[57] Juan Carlos Saez, Daniel Shelepov, Alexandra Fedorova, and Manuel Prieto. 2011. Leveraging workload diversity through OS scheduling to maximize performance on single-ISA heterogeneous multicore systems. *J. Parallel and Distrib. Comput.* 71, 1 (2011), 114–131. doi:10.1016/j.jpdc.2010.08.020

[58] Lars Schor, Iuliana Bacivarov, Devendra Rai, Hoeseok Yang, Shin-Haeng Kang, and Lothar Thiele. 2012. Scenario-Based Design Flow for Mapping Streaming Applications onto on-Chip Many-Core Systems. In *Proceedings of the 2012 International Conference on Compilers, Architectures and Synthesis for Embedded Systems* (Tampere, Finland) *(CASES '12)*. Association for Computing Machinery, New York, NY, USA, 71–80. doi:10.1145/2380403.2380422

[59] Lars Schor, Iuliana Bacivarov, Hoeseok Yang, and Lothar Thiele. 2014. AdaPNet: Adapting process networks in response to resource variations. In *2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. 1–10. doi:10.1145/2656106.2656112

[60] Andreas Schranzhofer, Jian-Jian Chen, and Lothar Thiele. 2010. Dynamic Power-Aware Mapping of Applications onto Heterogeneous MPSoC Platforms. *IEEE Transactions on Industrial Informatics* 6, 4 (2010), 692–707. doi:10.1109/TII.2010.2062192

[61] Robert Schöne, Thomas Ilsche, Mario Bielert, Andreas Gocht, and Daniel Hackenberg. 2019. Energy Efficiency Features of the Intel Skylake-SP Processor and Their Impact on Performance. In *International Conference on High Performance Computing & Simulation* (Dublin, Ireland, 2019-07) *(HPCS '19)*. IEEE, 399–406. doi:10.1109/HPCS48598.2019.9188239

[62] Hamid Shojaei, Twan Basten, Marc Geilen, and Azadeh Davoodi. 2013. A Fast and Scalable Multidimensional Multiple-Choice Knapsack Heuristic. *ACM Trans. Des. Autom. Electron. Syst.* 18, 4, Article 51 (oct 2013), 32 pages. doi:10.1145/2541012.2541014

[63] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[64] Amit Kumar Singh, Akash Kumar, and Thambipillai Srikanthan. 2013. Accelerating Throughput-Aware Runtime Mapping for Heterogeneous MPSoCs. *ACM Trans. Des. Autom. Electron. Syst.* 18, 1, Article 9 (jan 2013), 29 pages. doi:10.1145/2390191.2390200

[65] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. 2013. Mapping on Multi/Many-Core Systems: Survey of Current and Emerging Trends. In *Proceedings of the 50th Annual Design Automation Conference* (Austin, Texas) *(DAC '13)*. Association for Computing Machinery, New York, NY, USA, Article 1, 10 pages. doi:10.1145/2463209.2488734

[66] Till Smejkal, Marcus Hähnel, Thomas Ilsche, Michael Roitzsch, Wolfgang E. Nagel, and Hermann Härtig. 2017. E-Team: Practical Energy Accounting for Multi-Core Systems. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 589–601. https://www.usenix.org/conference/atc17/technical-sessions/presentation/smejkal

[67] Till Smejkal and Robert Khasanov. 2025. *Artifacts for HARP: Energy-Aware and Adaptive Management of Heterogeneous Processors.* doi:10.5281/zenodo.17370176

[68] Jan Spieck, Stefan Wildermann, and Jürgen Teich. 2022. A Learning-Based Methodology for Scenario-Aware Mapping of Soft Real-Time Applications onto Heterogeneous MPSoCs. *ACM Trans. Des. Autom. Electron. Syst.* 28, 1, Article 4 (dec 2022), 40 pages. doi:10.1145/3529230

[69] Kenzo Van Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvaez, and Joel Emer. 2012. Scheduling heterogeneous multi-cores through performance impact estimation (PIE). *ACM SIGARCH Computer Architecture News* 40, 3 (2012), 213–224.

[70] Adriano Vogel, Dalvan Griebler, Marco Danelutto, and Luiz Gustavo Fernandes. 2022. Self-adaptation on parallel stream processing: A systematic review. *Concurrency and Computation: Practice and Experience* 34, 6 (2022), e6759. doi:10.1002/cpe.6759

[71] Siqi Wang, Gayathri Ananthanarayanan, Yifan Zeng, Neeraj Goel, Anuj Pathania, and Tulika Mitra. 2020. High-Throughput CNN Inference on Embedded ARM Big.LITTLE Multicore Processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2020), 2254–2267. doi:10.1109/TCAD.2019.2944584

[72] Andreas Weichslgartner, Deepak Gangadharan, Stefan Wildermann, Michael Glaß, and Jürgen Teich. 2014. DAARM: design-time application analysis and run-time mapping for predictable execution in many-core systems. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis* (New Delhi, India) *(CODES '14)*. Association for Computing Machinery, New York, NY, USA, Article 34, 10 pages. doi:10.1145/2656075.2656083

[73] Stefan Wildermann, Michael Glaß, and Jürgen Teich. 2014. Multi-objective distributed run-time resource management for many-cores. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. doi:10.7873/DATE.2014.234

[74] Stefan Wildermann, Andreas Weichslgartner, and Jürgen Teich. 2015. Design Methodology and Run-Time Management for Predictable Many-Core Systems. In *2015 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. 103–110. doi:10.1109/ISORCW.2015.48

[75] Vasco Miguel Liang Xu, Liam White McShane, and Daniel Mossé. 2021. LUSH: Lightweight Framework for User-level Scheduling in Heterogeneous Multicores. In *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. 396–404. doi:10.1109/MCSoC51149.2021.00065

[76] Chantal Ykman-Couvreur, Vincent Nollet, Francky Catthoor, and Henk Corporaal. 2011. Fast multidimension multichoice knapsack heuristic for MP-SoC runtime management. *ACM Transactions on Embedded Computing Systems (TECS)* 10, 3, Article 35 (May 2011), 16 pages. doi:10.1145/1952522.1952528

[77] Teng Yu, Runxin Zhong, Vladimir Janjic, Pavlos Petoumenos, Jidong Zhai, Hugh Leather, and John Thomson. 2021. Collaborative Heterogeneity-Aware OS Scheduler for Asymmetric Multicore Processors. *IEEE Transactions on Parallel and Distributed Systems* 32, 5 (2021), 1224–1237. doi:10.1109/TPDS.2020.3045279

[78] Heechul Yun, Renato Mancuso, Zheng-Pei Wu, and Rodolfo Pellizzoni. 2014. PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 155–166. doi:10.1109/RTAS.2014.6925999

[79] Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. 2013. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 55–64. doi:10.1109/RTAS.

2013.6531079
[80] Sergey Zhuravlev, Juan Carlos Saez, Sergey Blagodurov, Alexandra Fedorova, and Manuel Prieto. 2012. Survey of Scheduling Techniques for Addressing Shared Resources in Multicore Processors. *ACM Comput. Surv.* 45, 1, Article 4 (dec 2012), 28 pages. doi:10.1145/2379776.2379780